

# Tecnología de vanguardia , Sugerencias y trucos para ListView

Dino Esposito

En la entrega del mes pasado, le introdujimos al control ListView, una adición al cuadro de herramientas de control de ASP.NET 3.5. Para recapitular, ListView es una versión mejorada del control DataList que proporciona más control sobre el marcado generado, la compatibilidad con la paginación y la integración completa con el modelo de enlace basado en el origen de datos.

En esta columna, profundizaré en las plantillas de ListView y el enlace de datos para implementar algunas características que son bastante comunes en las páginas del mundo real pero que requieren codificación adicional. Aprenderá a usar controles ListView anidados para crear vistas jerárquicas de datos y a ampliar el modelo de sistema de eventos de ListView derivando una clase personalizada de ListView.

En particular, refinaré el modelo de eventos para que pueda usar una plantilla diferente para los distintos grupos de elementos de datos enlazados. Por ejemplo, podrá usar una plantilla diferente para todos los elementos de datos incluidos en un conjunto de datos que coincidan con algunos criterios dados. Esto va mucho más allá de estilizar simplemente algunos elementos de forma distinta; eso ya lo puede hacer de forma sencilla en cualquier control de vista administrando simplemente el evento de ItemDataBound.

En general, los menús se implementan como una secuencia de etiquetas <li> que se estilizan con CSS. La representación de un menú plano no plantea ningún problemas concreto de enlace, ¿pero qué sucede cuando necesita un submenú o más? En este caso, puede usar el control integrado de menú o diseñar una estrategia más personalizada de representación usando un control ListView. Advierta que el control de menú usa una salida basada en tablas de forma predeterminada, a diferencia de la salida más adecuada para CSS que puede obtener con un control ListView. Para obtener una salida adecuada para CSS para un control de menú, necesita instalar y configurar el kit de herramientas denominado CSS Control Adapter Toolkit que puede descargar de [www.asp.net](http://www.asp.net).

## Creación de un menú jerárquico

Muchas aplicaciones web ofrecen un menú vertical a la izquierda o a la derecha de la página. Este menú permite al usuario navegar hasta páginas que se encuentran a dos o más niveles de anidamiento de profundidad. El control de menú de

ASP.NET es definitivamente una opción viable en este caso. Sin embargo, yo tiendo a usar el control de menú sólo cuando tengo un origen de datos jerárquico (normalmente un archivo XML) para alimentar el menú y cuando necesito crear submenús flotantes.

En el caso de las listas de elementos estáticas de varios niveles, prefiero usar un control tipo repetidor para dar salida al marcado creado por un equipo de diseño de IU. En ASP.NET 3.5, el control tipo repetidor preferido es ListView.

Considere un menú como el de la Figura 1. Se muestra en la plantilla HTML gratuita CoffeeNCream, que puede descargar de oswd.org. En la página de ejemplo, simplemente incorporé el marcado HTML en una página maestra de ASP.NET.



**Figura 1** Un menú estándar (Hacer clic en la imagen para ampliarla)

El código fuente HTML de un elemento del menú situado a la derecha tiene el siguiente aspecto:

```
<h1>Something</h1>
<ul>
  <li><a href="#">pellentesque</a></li>
  <li><a href="#">sociis natoque</a></li>
  <li><a href="#">semper</a></li>
  <li><a href="#">convallis</a></li>
```

</ul>

Como puede ver, contiene una cadena de nivel superior seguida por una lista de vínculos. Usa un primer control ListView para crear los elementos H1 y, a continuación, un ListView anidado (o un control semejante enlazado a datos) para representar la lista de vínculos. El primer paso es obtener los datos para rellenar el menú. De manera ideal, usaría una colección de los siguientes objetos de pseudotipo para generar cada elemento:

```
class MenuItem {
    public string Title;
    public Collection<Link> Links;
}

class Link {
    public string Url;
    public string Text;
}
```

Un método razonable para rellenar una colección de MenuItem es representar información a partir de un archivo XML. Aquí tiene un posible esquema para el documento:

```
<Data>
  <RightMenuItems>
    <MenuItem>
      <Title>Something</Title>
      <Link url="..." text="pellentesque" />
      :
    </MenuItem>
  </RightMenuItems>
</Data>
```

A continuación se demuestra cómo usar LINQ to XML para cargar y procesar el contenido:

```
var doc = XDocument.Load(Server.MapPath("dataMap.xml"));
var menu = (from e in doc.Descendants("RightMenuItems")
            select e).First();
var menuLinks = from mi in menu.Descendants("MenuItem")
                select new
                {
                    Title = mi.Value,
                    Links = (...)
                };
```

Después de cargar el documento, seleccione el primer nodo denominado RightMenuItems y, a continuación, obtenga todos los elementos secundarios de MenuItem. El contenido de cada nodo de MenuItem se carga en un nuevo tipo

anónimo con dos propiedades: Title y Links. ¿Cómo rellenaría la colección de Links? Aquí tiene algún código:

```
Links = (from l in mi.Descendants("Link")
        select new {Url=l.Attribute("url").Value,
                   Text=l.Attribute("text").Value})
```

El paso siguiente consiste en enlazar estos datos compuestos a la interfaz de usuario. Como hemos mencionado, se usa un control ListView exterior para representar el título y un segundo ListView anidado para representar la lista de vínculos de elemento secundarios (consulte la [Figura 2](#)). Tenga en cuenta que el ListView más interno se debe enlazar a datos mediante el método Eval, ya que cualquier otro enfoque no funcionará:

```
<asp:ListView runat="server" ID="subMenu"
  ItemPlaceholderID="PlaceHolder3"
  DataSource='<%# Eval("Links") %>'>
  ...
</asp:ListView>
```

El proceso de enlace a datos se inicia adjuntando datos al control ListView de nivel superior. Cuando esto sucede, el cuerpo del ListView se representa completamente, incluyendo el ListView anidado. Usted puede, teóricamente, interceptar el evento de ItemDataBound del ListView primario, recorrer el árbol de controles, obtener una referencia al ListView secundario y enlazarlo a datos mediante programación. Si lo hace, no generará una excepción, pero se perderá el comando de enlace del ListView interior, al desencadenarse demasiado tarde como para afectar a la representación. Una expresión de enlace a datos, por otra parte, se evalúa automáticamente durante cualquier evento de enlace a datos en el punto exacto del ciclo de vida del control. Esto asegura que los datos correctos se enlazan apropiadamente a la interfaz de usuario.

### Creación de una vista jerárquica

El mismo modelo que se usa para rellenar un menú jerárquico se puede emplear para crear alguna vista jerárquica de datos. En este caso, una opción alternativa sería usar un control TreeView para preparar una representación de varios niveles de los datos. Sin embargo, el enlace a datos en un control TreeView requiere un origen de datos jerárquico. Usar controles ListView anidados le da más flexibilidad con respecto a la estructura del origen de datos y a la interfaz de usuario resultante. Elaboremos un poco estos conceptos.

Suponga que necesita crear una cuadrícula jerárquica de datos en la que los clientes, los pedidos y los detalles de los pedidos se muestran en función de las relaciones de tabla existentes. ¿Cómo recuperaría los datos y los enlazaría al

control? Observe el código de la [Figura 3](#). Puede usar LINQ to SQL para cargar datos de forma sencilla en un modelo de objetos que se presta naturalmente a contener una jerarquía de datos. Tenga en cuenta que cuando ejecuta una consulta en LINQ to SQL, sólo recupera los datos que solicita explícitamente. Dicho de otra manera, sólo se captura el primer nivel del gráfico. Otros objetos relacionados no se cargan automáticamente al mismo tiempo.

La clase `DataLoadOptions` se puede usar para modificar el comportamiento predeterminado del motor de LINQ to SQL para que se carguen inmediatamente los datos a los que hace referencia una relación específica. El código de la [Figura 3](#) asegura que los pedidos cargados con clientes y detalles se cargan junto con pedidos.

El método `LoadWith` carga los datos según la relación especificada. Entonces, el método `AssociateWith` puede permitir el filtrado de objetos precapturados relacionados, como se muestra aquí:

```
opt.AssociateWith<Customer>(
    c => c.Orders.Where(o => o.OrderDate.Value.Year == 1997));
```

En este ejemplo, sólo se precapturan los pedidos emitidos en el año 1997 al capturar datos del cliente. Se usa el método de `AssociateWith` cuando se necesita precapturar datos relacionados y cuando se necesita aplicar un filtro. Depende enteramente de usted asegurar que no existen referencias cíclicas entre tablas; por ejemplo, cuando carga los pedidos para un cliente y, a continuación, carga el cliente para un pedido, como se muestra aquí:

```
DataLoadOptions opt = new DataLoadOptions();
opt.LoadWith<Customer> (c => c.Orders);
opt.LoadWith<Order> (o => o.Customer);
```

Ahora que ya tiene todos los datos preparados, puede empezar a pensar en el enlace. En este caso, un control `ListView` de dos niveles funcionará bastante bien. Enlaza el `ListView` de nivel superior a la colección de objetos `Customer` y el `ListView` más interno a la propiedad `Orders` de cada objeto `Customer` enlazado. El código de la [Figura 4](#) muestra el marcado correspondiente a una vista jerárquica de tres niveles en la que los clientes se muestran en el primer nivel y están representados por la propiedad `ItemTemplate` del `ListView` más exterior. El `ListView` incrustado se enlaza entonces a los pedidos. Finalmente, la propiedad `ItemTemplate` del control `ListView` incrustado contiene un `GridView` para enumerar los detalles de cada pedido.

Mejore la experiencia de usuario con extensores

Francamente, la interfaz de usuario que obtiene del código de la [Figura 4](#) no es muy atractiva. Puesto que se está construyendo una vista jerárquica de datos, un panel de expandir o contraer sería una solución especialmente apta para mejorar la experiencia de usuario. El kit de herramientas denominado ASP.NET AJAX Control Toolkit proporciona un extensor preparado que al aplicarse a un control de servidor de panel, puede agregar un efecto desplegable a la información asociada a cada cliente y pedido.

Use el control `CollapsiblePanelExtender` para definir un panel en el árbol de controles de página que se ampliará y contraerá mediante un script. No hace falta decir que como desarrollador de páginas no tendrá que escribir JavaScript. El control extensor inserta automáticamente todo el script necesario para ampliar y contraer el panel. Echemos un vistazo a las propiedades que quizás desee configurar en el extensor:

```
<act:CollapsiblePanelExtender runat="server" ID="CollapsiblePanel1"
  TargetControlID="panelCustomerDetails"
  Collapsed="true"
  ScrollContents="true"
  SuppressPostBack="true"
  ExpandedSize="250px"
  ImageControlID="Image1"
  ExpandedImage="~/images/collapse.jpg"
  CollapsedImage="~/images/expand.jpg"
  ExpandControlID="Image1"
  CollapseControlID="Image1">
</act:CollapsiblePanelExtender>
```

Es necesario realizar algunos cambios menores al código de la [Figura 4](#) para admitir el extensor de panel contraíble. En particular, debería editar el panel denominado `panelCustomerInfo` para agregar el botón que se usa para ampliar y contraer la vista secundaria. Aquí tiene una forma de rescribir el marcado del panel:

```
<asp:Panel ID="panelCustomerInfo" runat="server">
  <div class="customerInfo">
    <div style="float: left;"><%# Eval("CompanyName") %></div>
    <div style="float: right; vertical-align: middle;">
      <asp:ImageButton ID="Image1" runat="server"
        ImageUrl="~/images/expand.jpg"
        AlternateText="(Show Orders...)" />
    </div>
  </div>
</asp:Panel>
```

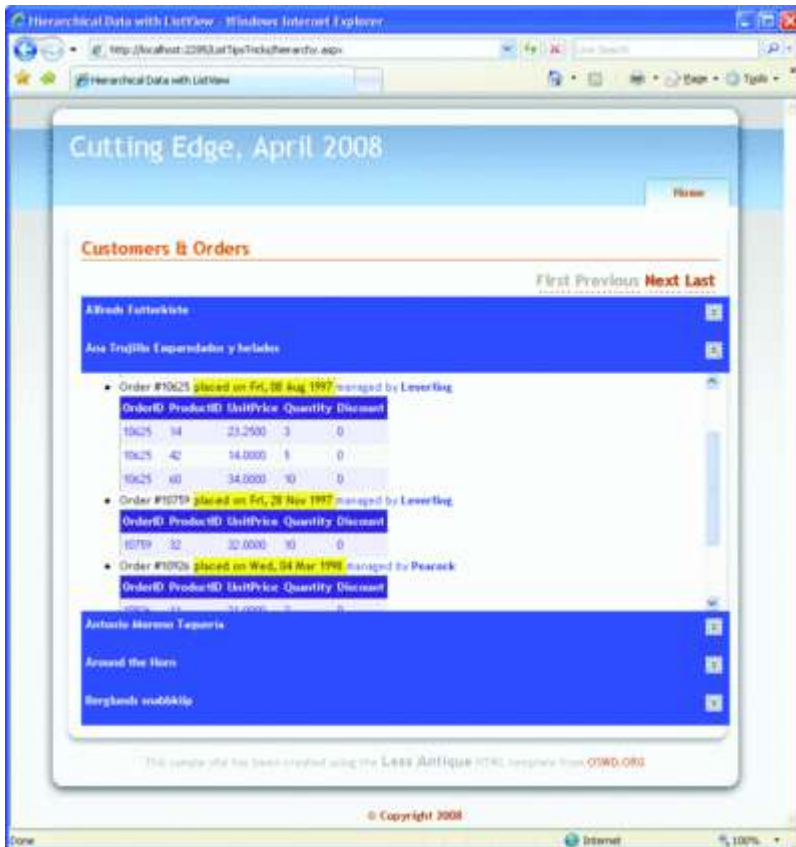
El botón se representa usando una imagen justificada a la derecha en la misma fila que el nombre del cliente. La propiedad `TargetControlID` del extensor hace referencia el panel de la página que se contraerá y ampliará. Éste es el panel que

contiene físicamente los pedidos y sus detalles. Como puede ver en la [Figura 4](#), es el panel denominado `panelCustomerDetails`.

Los atributos `ExpandControlID` y `CollapseControlID` indican los identificadores de los elementos que amplían y contraen el panel de destino cuando se hace clic en ellos. Si planea usar imágenes diferentes para reflejar el estado del panel, necesitará especificar también el identificador de un control de imagen. Esta información pertenece al atributo `ImageControlID`. El atributo `ImageControlID` está asociado a otras dos propiedades, `CollapsedImage` y `ExpandedImage`, que contienen la dirección URL de las imágenes.

La propiedad `ExpandedSize` establece la altura máxima en píxeles permitida para el panel ampliado. De forma predeterminada, se descarta cualquier contenido que exceda la altura. Sin embargo, si establece la propiedad `ScrollContents` en `true`, se agrega una barra de desplazamiento vertical para permitir que los usuarios desplacen el contenido entero.

Finalmente, la propiedad `Collapsed Boolean` le permite establecer el estado inicial del panel y `SuppressPostBack` indica si la expansión del panel debe ser una operación desarrollada completamente en el lado del cliente. Si se establece `SuppressPostBack` en `true`, no se usará ninguna devolución para ampliar o contraer el panel. Esto significa que no se pueden realizar actualizaciones sobre los datos mostrados. Para los datos relativamente estáticos que no cambian con frecuencia, ésta es definitivamente la mejor elección posible, ya que reduce el parpadeo en la página y el tráfico de red. Sin embargo, si necesita mostrar los datos de manera más dinámica en el control, puede minimizar el parpadeo mediante un control `UpdatePanel`. La [Figura 5](#) muestra la interfaz de usuario resultante de una vista de datos en tres niveles.



**Figura 5** Vista de datos en tres niveles (Hacer clic en la imagen para ampliarla)

## DataPager y ListView

El control ListView proporciona capacidad de paginación a través del nuevo control DataPager. DataPager es un control de paginación de uso general que puede ser usado por cualquier control enlazado a datos que implemente la interfaz IPageableItemContainer. A partir de ASP.NET 3.5, ListView es el único control que admite esta interfaz.

El control de DataPager puede mostrar una interfaz de usuario integrada o basada en plantillas. Siempre que el usuario haga clic para saltar a una página nueva, el control DataPager invoca un método en la interfaz IPageableItemContainer. Se espera que este método establezca las variables internas del control paginado para que sólo se muestre una página específica de datos durante la siguiente operación de enlace a datos.

Resulta que seleccionar la página de datos correcta sigue siendo el problema del control enlazado a datos (en este caso, ListView). Como ocurre con otros controles de "vista" en ASP.NET, el control ListView depende de código externo para paginar. Si los datos se enlazan a través de la propiedad de origen de datos, el código de usuario debe proporcionar los datos paginados. Por el contrario, si los

datos se enlazan a través de un control de origen de datos, debería configurar el control de origen de datos apropiadamente para admitir la paginación.

Los controles `LinqDataSource` y `ObjectDataSource` ofrecen capacidades de paginación integradas. `LinqDataSource` tiene la propiedad `AutoPage` para habilitar o deshabilitar la paginación predeterminada. Para los datos jerárquicos, también necesita asegurarse de que el contexto de datos LINQ tiene las opciones apropiadas de carga configuradas. La interfaz de programación de `LinqDataSource` no dispone de las propiedades para la configuración de la propiedad `LoadOptions` en el objeto del contexto de datos. Sin embargo, mediante la administración del evento `ContextCreated`, puede tener acceso al contexto de datos recién creado y configurarlo como quiera:

```
void LinqDataSource1_ContextCreated(
    object sender, LinqDataSourceStatusEventArgs e)
{
    // Get a reference to the data context
    DataContext db = e.Result as DataContext;

    if (db != null)
    {
        DataLoadOptions opt = new DataLoadOptions();
        opt.LoadWith<Customer>(c => c.Orders);
        opt.LoadWith<Order>(o => o.Employee);
        opt.LoadWith<Order>(o => o.Order_Details);
        db.LoadOptions = opt;
    }
}
```

Como alternativa a esta acción, puede usar el control `ObjectDataSource` para proporcionar datos e implementar cualquier lógica de paginación. Entonces, en el objeto empresarial puede usar LINQ to SQL o ADO.NET sin formato para el acceso a datos.

Vale la pena mencionar, sin embargo, un problema con el me encontré al usar `DataPager` y `ListView` juntos. Inicialmente tenía una página de contenidos con `ListView` y `DataPager` hospedados en el mismo marcador de posición de contenidos. Hice referencia al control `ListView` del control `DataPager` usando la propiedad `PagedControlID`, como se muestra a continuación. Funcionó de maravilla:

```
<asp:DataPager ID="DataPager1" runat="server"
    PagedControlID="ListView1"
    PageSize="5"
    EnableViewState="false">
<Fields>
    <asp:NextPreviousPagerField
        ShowFirstPageButton="true"
```

```
ShowLastPageButton="true" />  
</Fields>  
</asp:DataPager>
```

A continuación, moví DataPager a otra área de contenidos de la misma página maestra. De repente, DataPager dejó de comunicarse con el control ListView. El problema reside en el algoritmo usado por el control DataPager para localizar el control paginado. Este algoritmo no funciona si los dos controles se hospedan en dos contenedores de nomenclatura diferentes. Para solucionar el problema, necesita identificar el control paginado usando su identificador completo y único (que incluye información del contenedor de nomenclatura). Por desgracia, no puede establecer fácilmente esta información de forma declarativa.

No puede usar bloques de código de estilo ASP porque se tratan como literales al usarse para establecer una propiedad de un control de servidor. No puede usar una expresión de enlace a datos <%#... %> porque la expresión se evalúa demasiado tarde para las necesidades del control DataPager. El evento Load tiene lugar demasiado tarde y causaría que el control DataPager generara una excepción. La solución alternativa más sencilla consiste en establecer la propiedad PagedControlID mediante programación en el evento Init de la página, así:

```
protected void Page_Init(object sender, EventArgs e)  
{  
    DataPager1.PagedControlID = ListView1.UniqueID;  
}
```

## Varias plantillas de elementos

Como ocurre con otros controles basados en plantillas o enlazados a datos, el control ListView repite la misma plantilla de elementos para cada elemento de datos enlazado. ¿Qué ocurre si desea cambiarlo por un cierto subconjunto de elementos? Honestamente, tengo que admitir que nunca he tenido la necesidad de usar más de una plantilla de elementos en todos los años que he pasado programando ASP.NET. Varias veces he personalizado la apariencia de un grupo pequeño de elementos en los controles DataGrid y GridView en función de las condiciones de tiempo de ejecución. Sin embargo, eso siempre trajo consigo la aplicación de un conjunto diferente de atributos de estilo.

Sólo en muy contadas ocasiones agregué nuevos controles mediante programación (en su mayoría controles Label o celdas de tabla) a la plantilla existente. Esta tarea no es complicada cuando se trata de controles enlazados a datos que generan eventos de enlace a datos (al menos no si tiene un conocimiento profundo de la estructura interna de los controles que manipula).

Aunque la inserción de controles mediante programación es una solución que funcionó realmente bien en la práctica, nunca acabó de cautivarme. Así que decidí

tirar por un camino diferente cuando un cliente me pidió que modificara un menú basado en ListView en una página web. En un menú semejante al de la Figura 1, tuve que representar los elementos de un submenú de manera horizontal en vez de vertical.

El control ListView genera su marcado pasando por el origen de datos y aplicando el siguiente algoritmo. Primero, comprueba si se requiere un separador de elementos. Si éste es el caso, crea una instancia de plantilla y el objeto de elemento de datos. El objeto del elemento de datos es el contenedor de la plantilla de elementos y alberga información acerca del índice del elemento en el origen de la vista y de los datos enlazados. Cuando se crea la instancia de plantilla de elementos, se genera el evento ItemCreated. El paso siguiente es el enlace a datos. Una vez completada esta acción, se genera el evento ItemDataBound.

Como puede ver, no hay ningún evento público administrable que le permita cambiar la plantilla para cada elemento mediante programación. Puede cambiar la plantilla en los eventos de página Load e Init, aunque esto afectaría a todos los elementos enlazados. Si administra ItemCreated para establecer ahí la propiedad ItemTemplate, el cambio afectará al elemento siguiente, pero no al elemento que se está procesando actualmente. Necesitaría un evento ItemCreating, pero el control ListView no genera este tipo de evento. La solución, entonces, consiste en crear su propio control ListView, como en la [Figura 6](#).

El reemplazo del método CreateDataItem, le da la oportunidad de ejecutar el código justo antes de que se cree la instancia de plantilla de elementos. El método CreateDataItem se declara protegido y virtual en la clase ListView. Como puede ver en la [Figura 6](#), el reemplazo del método es bastante sencillo. Primero genera un evento ItemCreating personalizado y, a continuación, llama al método base.

El evento ItemCreating devuelve un par de enteros al código de usuario (el índice absoluto del elemento del origen de datos y el índice específico de la página). Por ejemplo, para un tamaño de página de 10, cuando el control ListView trabaja para representar el primer elemento de la segunda página, dataIndex y displayIndex contienen 11 y 1 elementos respectivamente. Para usar el nuevo evento ItemCreating, declare simplemente el método y el controlador en el control ListView personalizado, como puede ver en el código siguiente:

```
<x:ListView runat="server" ID="ListView1"
  ItemPlaceholderID="itemPlaceholder"
  DataSourceID="ObjectDataSource1"
  OnItemCreating="ListView1_ItemCreating">
  <LayoutTemplate>
    <div>
      <asp:PlaceHolder runat="server" ID="itemPlaceholder" />
    </div>
  </LayoutTemplate>
```

</x:ListView>

En el código, puede administrar el evento de la siguiente manera:

```
void ListView1_ItemCreating(  
    object sender, ListViewItemCreatingEventArgs e)  
{  
    string url = "standard.ascx";  
    if (e.DisplayIndex % DataPager1.PageSize == 0)  
        url = "firstItem.ascx";  
  
    ListView1.ItemTemplate = Page.LoadTemplate(url);  
}
```

Aquí se emplean dos controles de usuario diferentes para representar los elementos de datos. El índice de visualización determina el control de usuario específico. Todos elementos menos el primero comparten la misma plantilla. La Figura 7 muestra la página en acción.



**Figura 7** Varias plantillas de elementos (Hacer clic en la imagen para ampliarla)

Cuando piensa en la complejidad de las páginas comunes del mundo real, esta solución aparece demasiado sencilla. La mayoría de las veces, necesita usar plantillas diferentes en función del contenido que vaya a mostrar. Necesita mejorar

aún más el control ListView personalizado para cambiar la plantilla de elementos dentro del proceso de enlace a datos. Observe el código de la [Figura 8](#).

El método CreateDataItem genera el evento ItemCreating y almacena en caché el índice de visualización para un uso posterior. Además, reemplaza el método InstantiateItemTemplate para demorar la creación de la instancia de plantilla. Para este propósito se usa un indicador booleano privado. Como he mencionado anteriormente, el control ListView inicia el proceso de enlace a datos después de crear una instancia de plantilla de elementos.

Sin embargo, en la implementación mostrada en el código de la [Figura 8](#), no se crea ninguna instancia de plantilla de elementos hasta que se genera el evento ItemCreated. Cuando se provoca el evento ItemCreated, el objeto de elemento de datos se enlaza al contenedor del elemento de ListView a través de la propiedad DataItem. Administrando el evento ItemCreated en su código, puede decidir qué plantilla de elementos se va a usar en función del elemento de datos enlazado, como puede ver a continuación:

```
protected override void OnItemCreated(ListViewItemEventArgs e)
{
    base.OnItemCreated(e);

    _shouldInstantiate = true;
    InstantiateItemTemplate(e.Item, _displayIndex);
}
```

En este caso, el método base genera el evento ItemCreated para la página. A continuación, el control ListView personalizado restablece el indicador booleano e invoca el método para crear una instancia de plantilla de elementos. Por último, se crea una instancia de plantilla de elementos un poco más tarde que en el control ListView integrado, pero puede establecer mediante programación la propiedad ItemTemplate para cada elemento en el controlador de eventos ItemCreated después de mirar al contenido del elemento de datos enlazado (consulte la [Figura 9](#)). La [Figura 10](#) muestra una página de ejemplo en la que se usa una plantilla azul para hombres y una plantilla rosa para mujeres.



**Figura 10** Un menú estándar (Hacer clic en la imagen para ampliarla)

Resumiendo

En resumidas cuentas, el nuevo control ListView de ASP.NET 3.5 es una versión renovada del control DataList que conocemos desde ASP.NET 1.0. El control ListView permite un control más férreo sobre el marcado generado y admite completamente los objetos de origen de datos.

En esta columna, le hemos mostrado cómo usar controles ListView anidados para crear vistas de datos paginables y de varios niveles. También le hemos enseñado a modificar el proceso estándar de representación de ListView mediante la derivación de un control personalizado y el reemplazo de algunos métodos. El resultado final es un control que admite varias plantillas de elementos. Ningún otro control de enlace a datos de ASP.NET ofrece este nivel de flexibilidad.

Envíe sus preguntas y comentarios para Dino a la dirección [cutting@microsoft.com](mailto:cutting@microsoft.com).

---

**Dino Esposito** es arquitecto de IDesign y el autor de *Programming ASP.NET 3.5 Core Reference*. Con residencia en Italia, Dino participa habitualmente en conferencias y eventos del sector en todo el mundo. Puede ponerse en contacto con él en [cutting@microsoft.com](mailto:cutting@microsoft.com) o participar en su blog en [weblogs.asp.net/despos](http://weblogs.asp.net/despos).

---

© 2007 Microsoft Corporation and CMP Media, LLC. Reservados todos los derechos; queda prohibida la reproducción parcial o total sin previa autorización.